151228619/151248619: OBJECT ORIENTED PROGRAMMING I
2022-2023 FALL SEMESTERS
LAB FINAL REPORT

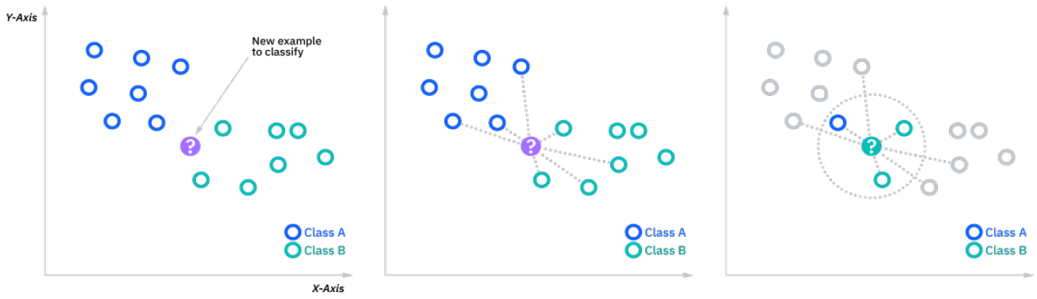
Name: Burak

Surname: KAYKAÇ

Student No: 151220174025

This c++ code is an example of knn algorithm.

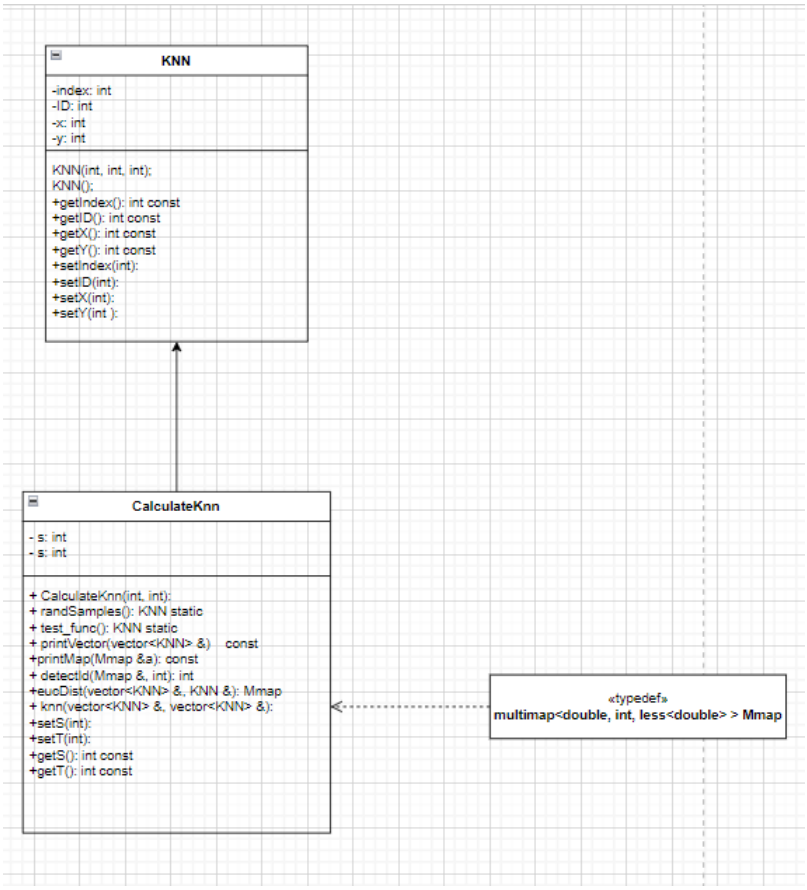
The k-nearest neighbor algorithm, also known as KNN or k-NN, is a non-parametric, supervised learning classifier that uses proximity to make classifications or predictions about the grouping of an individual data point. Although it can be used for both regression and classification problems, it is typically used as a classification algorithm that operates on the assumption that similar points can be found close together. ***(https://www.ibm.com/topics/knn)***



(KNN Algorithm)

WORKING PRINCIPLE

The uml diagram of the code is given below:



It has 2 classes and these classes provide to calculate and keep the information of the data. The KNN class has 4 private integer variables: index, ID, x and y. The variable 'x' holds the x component of the point on the plane, and the variable 'y' holds the y component of the point on the plane. The 'ID' variable has the information of which category that point belongs to. The 'index' variable has the information on which index that point is. set and get functions are created for each of these variables.

There are two constructors and one destructor. The overloaded constructor that assigns the values it receives to the index, x and y variables, respectively. and empty constructor.

The CalculateKnn class has 2 private variables, s and t. The variable 's' contains the number of sample vectors. The variable 't' contains the number of test vectors. 'multimap<double, int, less<double> > Mmap;' for use in this class typedef has been made. set and get functions are creted for the private members of this class

The constructor of this class takes 2 integer values, a and b, and assigns them to the variables s and t, which are private members , with the initilaze list method. All functions and assignments required for simplicity in the driver code are made here. First, the user is asked whether to activate the 'srand()' function to decide whether the values to be assigned are time

dependent. Then, the a and b values given to the constructor, as many as the sample and test vectors are created with the 'generate()' function. The sample vectors whose coordinates are randomly assigned, the first 10 samples having ID0, the next 10 samples having ID1 and the last 10 samples having ID2 (when s is set to 30). Creats 'b' test vectors whose ID number is unknown (assigned 43 invalid) whose positions are randomly determined. then the 'knn(vector<KNN> &samples ,vector<KNN> &test)' function is called to run the knn algorithm. After this process, the test group samples whose IDs are determined are added to the sample vector with the insert function. and in all these stages, vectors are displayed in the terminal with the 'printVector(vector<KNN> &a)' function.

The **'knn(vector<KNN> &samples ,vector<KNN> &test)'** function first asks the user for a value to determine the number of neighborhoods. then by looping up to 't' value, it measures the euclidean distance between the points with the 'eucDist(vector<KNN> &a, KNN &b)' function and assigns it to the map of type Mmap, then with the 'detectID(Mmap &a,int k)' function, it determines the Id by looking at the neighborhoods of this map and the test vector with that parameter Assigns the Id to the data.

The **'eucDist(vector<KNN> &a, KNN &b)'** function is a function that calculates Euclidean distances and keeps these distances in a 'Mmap' type map. calculates the eucludean distance with the following formula.

$$d(x,y) = \sqrt{\sum_{i=1}^{n}(y_i - x_i)^2}$$

The **'detectID(Mmap &a,int k)'** function returns an ID number as 0,1 or 2 by looking at the IDs of the nearest k neighborhoods according to the map and k value it receives. The reason for using map in these functions is avoid to make an extra sorting, because due to the map structure, when Euclidean distances are assigned to the key value, it automatically performs this ordering.

The **'printMap(Mmap &a)' f**unction is basically created to print data of the type 'Mmap' used in the terminal.

The **'printVector(vector<KNN> &a)'** function is basically created to print the 'KNN' type data used in the terminal.

The **'randSamples()'** function has been prepared to work in harmony with the 'generate' function of the stl library. The 'x' and 'y' coordinates of the test vector product were randomly determined between -5 and 5. ID parameter is assigned as undefined value 43. The index parameter is assigned in order from 30 to 39.

The **'test_func()'** function is used in harmony with stl's 'generate' function by randomly assigning 'x' and 'y' values between -5 and 5 as desired. and ID numbers are the first 10 data 0 the next 10 data 1, the last10 data 2.